Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное
учреждение высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»
Институт естественных и точных наук
Факультет математики, механики и компьютерных технологий
Кафедра прикладной математики и программирования

Сравнение быстродействия методов сортировки, быстрой сортировки Хоара и метода пузырька

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К КУРСОВОЙ РАБОТЕ
по дисциплине «Алгоритмы и структуры данных»
ЮУрГУ–010302.2020.153.ПЗ КР

Автор работы,
студент группы ЕТ-212
_____ / Т.С. Шерстобитов
«____» _____ 2020 г.

Руководитель работы,
старший преподаватель
_____ / А.С. Шелудько
«____» _____ 2020 г.

Работа защищена с оценкой
_____
«____» _____ 2020 г.

Челябинск 2020

АННОТАЦИЯ

Шерстобитов Т.С. Сравнение быстродействия методов сортировки, быстрой сортировки Хоара и метода пузырька. – Челябинск: ЮУрГУ, ЕТ-212, 2020. – 32 с., 17 ил, библиогр. список – 6 наим., 3 прил.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Актуальность темы. Правильный выбор алгоритма сортировки позволяет значительно оптимизировать время работы программ. Фактические данные и их визуализация позволяют начинающим программистам выбрать более подходящий алгоритм сортировки.

Цель работы – Разработать программу, выполняющую сравнение быстродействия двух методов сортировки, быстрой сортировки Хоара и метода пузырька

Объект работы – Программа выполняющая сравнение алгоритмов сортировки

Результаты работы можно использовать в процессе последующего обучения в соответствии с учебным планом подготовки бакалавров по направлению «Прикладная математика и информатика»

1 ПОСТАНОВКА ЗАДАЧИ

Необходимо разработать программу, выполняющую сравнение быстродействия двух методов сортировки:

метод пузырька;

быстрая сортировка Хоара.

Сравнение выполнить методом статистических испытаний. Приблизительная схема сравнения времени сортировки целого массива из N чисел:

1.Генерировать случайный целочисленный массив A (функции random и randomise). Генерируемые числа принадлежат диапазону от 0 до 109-1.

2.Копировать массив A в массив B.

3.Выполнить сортировку массива B методом пузырька и вычислить потраченное время (используйте функцию GetLocalTime или timeGetTime. Описание функций ищите в интернет)

4.Копировать массив A в массив B.

5.Выполнить сортировку массива B с помощью быстрой сортировки Хоара и вычислить потраченное время

Повторять пункты 1-5 L раз.

Найти среднее время сортировки для каждого из методов.

6.Перейти к следующему значению N.

Программа должна получить достаточное количество точек для построения графиков зависимостей времени сортировки от размера массива.

Значения L,N и другие исходные данные вводятся на старте программы из входного файла.

Напечатать таблицы зависимостей и построить совмещённые графики зависимостей времени сортировки от размера массива в координатах: по оси x – log10 (N), по оси y – log10(время(ms)).

Для построения графиков подберите в интернете подходящую программу, например, http://soft.mydiv.net/win/download-Graph.html.

2 АЛГОРИТМ РЕШЕНИЯ

Программа выполняет сравнение двух алгоритмов сортировки.

– метод пузырька;

– быстрая сортировка Хоара.

Ниже на рисунке 2.1 и рисунке 2.2 представлены блок схемы данных алгоритмов.

Рисунок 2.1 Алгоритм сортировки метода пузырька

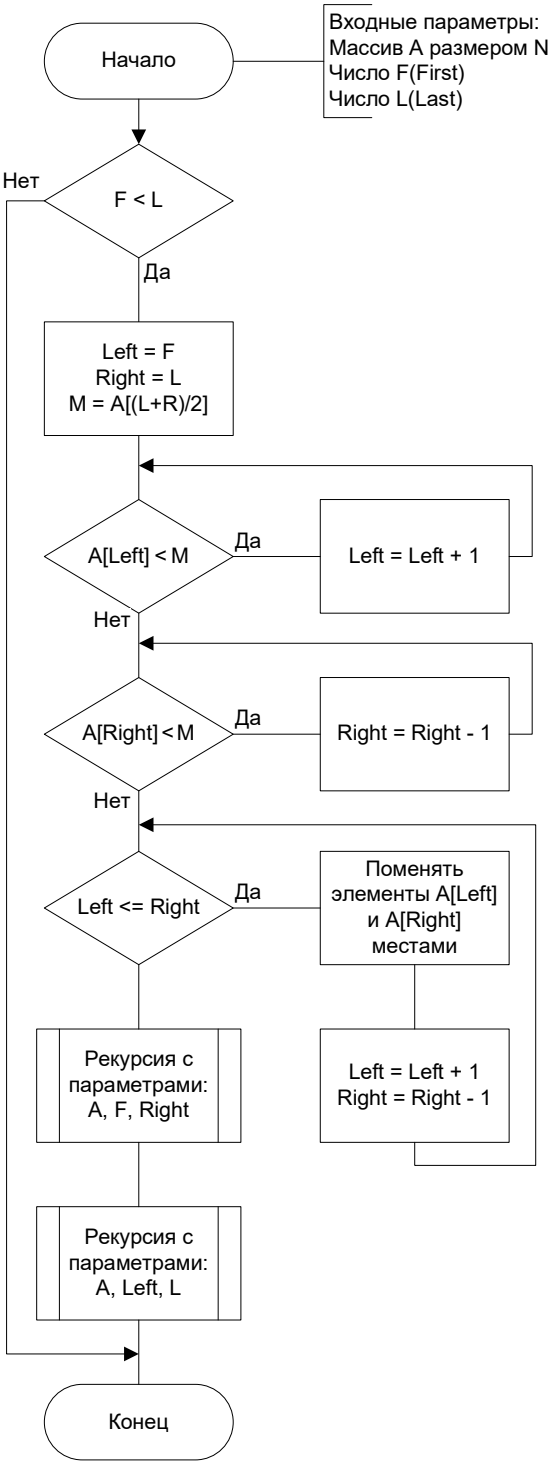

Рисунок 2.2 Алгоритм быстрой сортировки Хоара

Реализация алгоритма сортировки метода пузырька на языке программирования C++:

```
void bubbleSort(int* array, int n){
   for (int i = n - 1; i > 0; i--){
      for (int j = 0; j < i; j++){
         if (array[j] > array[j + 1]){
            int tmp = array[j];
            array[j] = array[j + 1];
            array[j + 1] = tmp;
         }
      }
   }
}
```

Реализация алгоритма быстрой сортировки Хоара на языке программирования C++:

```
void quickSort(int* array, int first, int last){
   if (first < last){
      int left = first;
      int right = last;
      int middle = array[(left + right) / 2];
      do{
         while (array[left] < middle){
            left++;
         }
         while (array[right] > middle){
            right--;
         }
         if (left <= right){
            int tmp = array[left];
            array[left] = array[right];
            array[right] = tmp;
            left++;
            right--;
         }
      } while (left <= right);
      quickSort(array, first, right);
      quickSort(array, left, last);
   }
}
```

Алгоритм выполнения сравнения алгоритмов сортировки приведён в рисунке 2.3.

Рисунок 2.3 Алгоритм выполнения сравнения алгоритмов сортировки

3 ОПИСАНИЕ ПРОГРАММЫ

Для выполнения сравнения методов сортировки необходимо разработать модуль выполняющий сравнение и содержащий данные о сравнении. Так как время выполнения быстрой сортировки Хоара в большинстве опытных случаев ровнялось 0 миллисекунд, было принято решение в качестве единицы измерения брать не секунды, а время процессора (тиках). Основываясь на алгоритме выполнения сравнения алгоритмов сортировки (см. рисунок 2.3), разработана интерфейсная часть программы. Ниже представлен рисунок 3.1, UML схема, содержащая минимально необходимые поля и методы, для проведения сравнения алгоритмов.

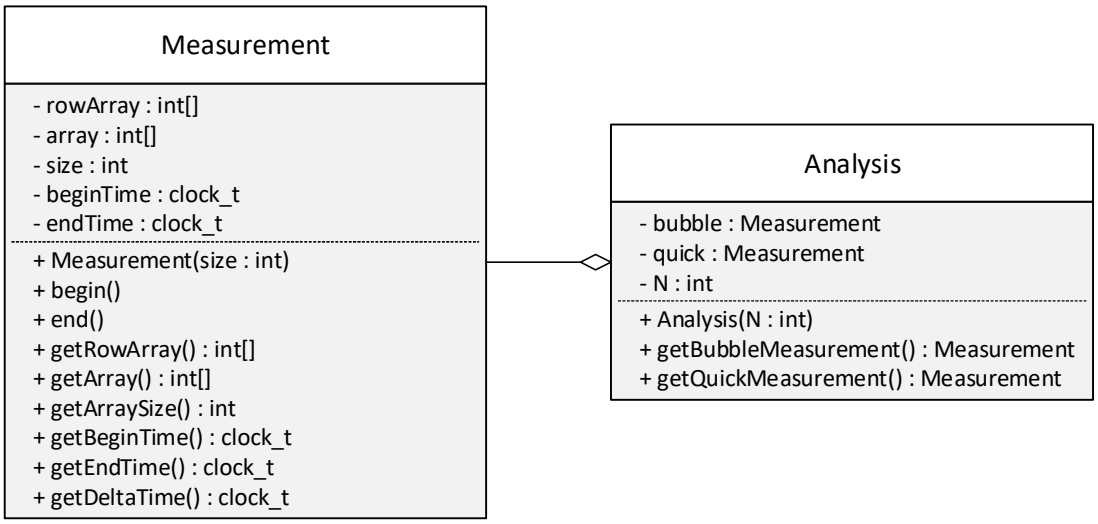

Рисунок 3.1 UML схема модуля Analysis

В соответствии со схемой (см. рисунок 2.3) была разработана интерфейсная часть модуля, представленная в файлах Analysis.hpp (см Приложение 1.3) и Measurement.hpp (см Приложение 1.4). Реализация интерфейсной части представлена в файлах Analysis.cpp (см Приложение 1.5) и Measurement.cpp (см Приложение 1.6).

Так как пользователю может быть удобна запись результата в различных вариантах, Необходимо так же разработать модуль позволяющий сохранить результаты сравнения в удобном для последующего применения формате.

Ниже представлен рисунок 2.5, UML схема, содержащая необходимые поля и методы, для реализации различных способов записи результатов сравнения алгоритмов.

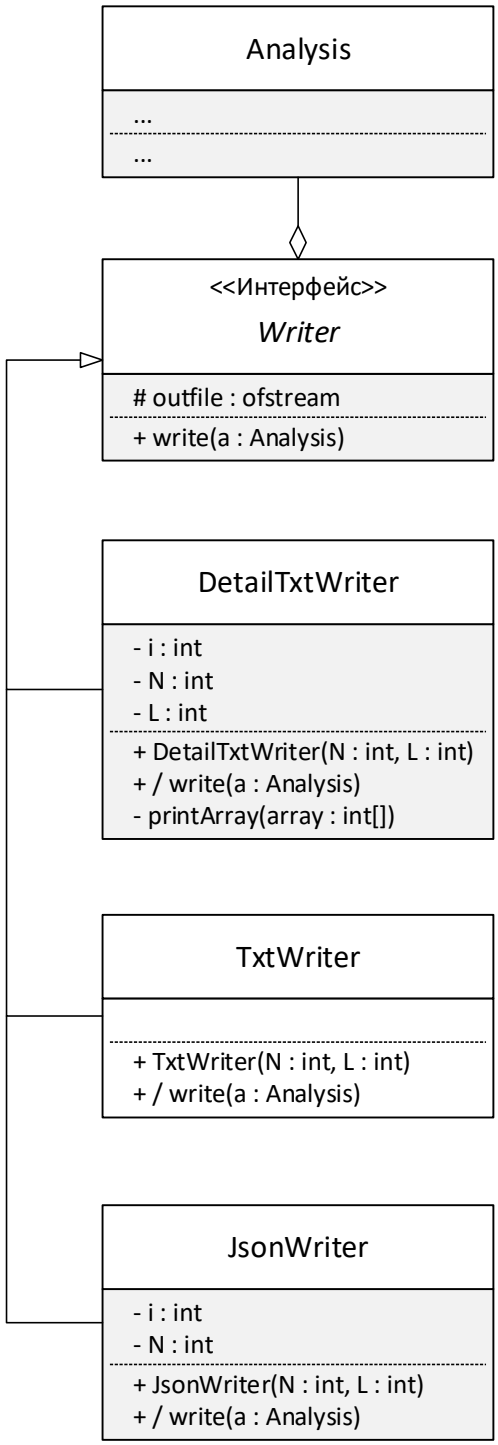

Рисунок 3.2 UML схема модуля Writer

В соответствии со схемой (см. рисунок 3.2) была разработана интерфейсная часть модуля, представленная в файле Writer.hpp (см Приложение 1.7). Реализация интерфейсной части представлена в файле Writer.cpp (см Приложение 1.8).

Разрабатываемая программа перед началом выполнения сравнения алгоритмов должна принять данные либо через ввод пользователя, либо через параметры командной строки. Различные виды получения данных, а так же запуск программы, реализованы в файле Main.cpp (см Приложение 1.9).

Для получения инструкций по использованию разрабатываемой программы см. Приложение 2.

Пример результатов выполнения программы см. Приложение 3.

ЗАКЛЮЧЕНИЕ

В результате работы была разработана программа выполняющая сравнение быстродействия двух методов сортировки:

метод пузырька;

быстрая сортировка Хоара.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Ахо, А.В. Структуры данных и алгоритмы / А.В. Ахо, Д.Э. Хопкрофт, Д.Д. Ульман. – М.: Вильямс, 2000. – 382 с.

2. Вирт, Н. Алгоритмы и структуры данных / Н. Вирт. – М.: Мир, 1989. – 360 с.

3. Кормен, Т. Алгоритмы: построение и анализ / Т. Кормен, Ч. Лейзерсон, Р. Ривест. – М.: МЦНМО, 2001. – 955 с.

4. Кнут, Д. Искусство программирования. Том 3. Сортировка и поиск / Д. Кнут. – М.: Вильямс, 2000. – 822 с.

5. Подбельский, В.В. Курс программирования на языке Си / В.В. Подбельский, С.С. Фомин. – М.: ДМК Пресс, 2012. – 384 с.

6. Хаггарти, Р. Дискретная математика для программистов / Р. Хаггарти. – М.: Техносфера, 2012. – 399 с.

ПРИЛОЖЕНИЕ 1

Текст программы

П1.1 Sort.hpp

```
#ifndef SORT_HPP
#define SORT_HPP

void bubbleSort(int* array, int n);
void quickSort(int* array,int first, int last);

#endif //MEASUREMENT_HPP
```

П1.2 Sort.cpp

```
#include "Sort.hpp"

void bubbleSort(int* array, int n){
   for (int i = n - 1; i > 0; i--){
      for (int j = 0; j < i; j++){
         if (array[j] > array[j + 1]){
            int tmp = array[j];
            array[j] = array[j + 1];
            array[j + 1] = tmp;
         }
      }
   }
}

void quickSort(int* array,int first, int last){
   if (first < last){
      int left = first, right = last, middle = array[(left +
right) / 2];
      do{
         while (array[left] < middle){
            left++;
         }
         while (array[right] > middle){
            right--;
         }
         if (left <= right){
            int tmp = array[left];
            array[left] = array[right];
            array[right] = tmp;
            left++;
            right--;
         }
      } while (left <= right);
      quickSort(array, first, right);
      quickSort(array, left, last);
   }
```

```
}
```

П1.3 Analysis.hpp

```
#include "Measurement.hpp"
#ifndef ANALYSIS_HPP
#define ANALYSIS_HPP

class Analysis{
   public:
      /**
       * Конструктор.
       * Число N - Размер массива для анализа
       */
      Analysis(unsigned int N);
      virtual ~Analysis();
      /**
       * Возвращает данные о работе алгоритма сравнения метода
пузырька
       */
      Measurement *getBubbleMeasurement() const;
      /**
       * Возвращает данные о работе алгоритма сравнения быстрой
сортировки Хоара
       */
      Measurement *getQuickMeasurement() const;
   private:
      Measurement *bubble, *quick;
      unsigned int N;
};

#endif //ANALYSIS_HPP
```

П1.4 Measurement.hpp

```
#include <ctime>
#ifndef MEASUREMENT_HPP
#define MEASUREMENT_HPP

/**
 * Даные о работе алгоритма
 */
class Measurement{
   public:
      /**
       * Создаёт новый объект класса данных о замерах,
       * с случайно сгенерированым массивом размера size.
       */
      Measurement(unsigned int size);
      Measurement(const Measurement &orig);
      virtual ~Measurement();
      /**
       * Установка точки начала замера
       */
      void begin();
      /**
       * Установка точки окончания замера
       */
      void end();
      /**
       * Возращает начальный массив;
       */
      int* getRowArray() const;
      /**
       * Возращает массив для сортировки
       */
      int* getArray() const;
      /**
       * Возращает размер массивов
       */
      unsigned int getArraySize() const;
      /**
       * Возращает время начала замера
       */
      clock_t getBeginTime() const;
      /**
       * Возращает время окончания замера
       */
      clock_t getEndTime() const;
      /**
       * Возращает время замера
       */
      clock_t getDeltaTime() const;
```

```
   private:
      int* rowArray;//Копия массива до сортировки
      int* array;//Массив для сортировки
      unsigned int size;//Размер массива
      clock_t beginTime = 0, endTime = 0;//Время начала/конца
замера
};

#endif //MEASUREMENT_HPP
```

П1.5 Analysis.cpp

```
#include "Analysis.hpp"
#include "Sort.hpp"

Analysis::Analysis(unsigned int N){
   bubble = new Measurement(N);
   quick = new Measurement(*bubble);
   bubble->begin();
   bubbleSort(bubble->getArray(), N);
   bubble->end();
   quick->begin();
   quickSort(quick->getArray(), 0, N - 1);
   quick->end();
}

Analysis::~Analysis(){
   delete bubble;
   delete quick;
}

Measurement *Analysis::getBubbleMeasurement() const{
   return bubble;
}

Measurement *Analysis::getQuickMeasurement() const{
   return quick;
}
```

П1.6 Measurement.cpp

```
#include "Measurement.hpp"
#include <random>
//Максимально возможное сгенерированное число
#define MAX 999999999

Measurement::Measurement(unsigned int size) : size(size) {
   static std::default_random_engine dre(time(0));
   array = new int[size];
   rowArray = new int[size];
   for (int i = 0; i < size; i++) {
      array[i] = rowArray[i] = dre() % MAX;
   }
}

Measurement::Measurement(const Measurement &orig){
   size = orig.size;
   array = new int[size];
   rowArray = new int[size];
   for (int i = 0; i < size; i++) {
      array[i] = orig.array[i];
      rowArray[i] = orig.array[i];
   }
}

Measurement::~Measurement() {
   delete[] array;
   delete[] rowArray;
}

void Measurement::begin() {
   beginTime = clock();
}

void Measurement::end() {
   endTime = clock();
}

int *Measurement::getRowArray() const {
   return rowArray;
}

int* Measurement::getArray() const{
   return array;
}

unsigned int Measurement::getArraySize() const {
   return size;
}
```

```
clock_t Measurement::getBeginTime() const {
   return beginTime;
}

clock_t Measurement::getEndTime() const {
   return endTime;
}

clock_t Measurement::getDeltaTime() const {
   return endTime - beginTime;
}
```

П1.7 Writer.hpp

```
#include "Analysis.hpp"
#include <fstream>
#ifndef WRITER_HPP
#define WRITER_HPP


class Writer{
   public:
      virtual ~Writer(){}
      virtual void write(Analysis *a) = 0;
   protected:
      std::ofstream outfile;
};

/*
 * Выводит информацию о замере в файл формата json
 */
class JsonWriter : public Writer{
   public:
      JsonWriter(unsigned int N, unsigned int L);
      ~JsonWriter();
      void write(Analysis *a);
   private:
      unsigned int i = 1, L;
};

/*
 * Выводит информацию о замере в файл формата txt
 */
class TxtWriter : public Writer{
   public:
      TxtWriter(unsigned int N, unsigned int L);
      ~TxtWriter();
      void write(Analysis *a);
};
```

```
/*
 * Выводит подробную информацию о замере в файл формата txt
 */
class DetailTxtWriter : public Writer{
   public:
      DetailTxtWriter(unsigned int N, unsigned int L);
      ~DetailTxtWriter();
      void write(Analysis *a);
   private:
      void printArray(int* array);
      unsigned int i = 1, L, N;
};

#endif //WRITER_HPP
```

П1.8 Writer.cpp

```
#include "Writer.hpp"
//void DetailTxtWriter::write(Analysis *a) -- CLOCKS_PER_SEC
#include <ctime>

JsonWriter::JsonWriter(unsigned int N, unsigned int L) : L(L){
   outfile.open(std::to_string(N) + "_" + std::to_string(L) +
".json");
   if(!outfile.is_open())
      exit(EXIT_FAILURE);
   outfile << "{\n\t\"N\" : " << N << ",\n\t\"L\" : " << L <<
",\n\t\"Measurements\": [\n";
}

JsonWriter::~JsonWriter(){
   outfile << "\t]\n}";
   outfile.close();
}
```

```
void JsonWriter::write(Analysis *a){
   outfile << "\t\t{ \"bubble\" : " << a-
>getBubbleMeasurement()->getDeltaTime()
         << ", \"quick\" : " << a->getQuickMeasurement()-
>getDeltaTime() << "}";
   if (i++ < L)
      outfile << ',';
   outfile << '\n';
}

TxtWriter::TxtWriter(unsigned int N, unsigned int L){
   outfile.open(std::to_string(N) + "_" + std::to_string(L) +
".txt");
   if(!outfile.is_open())
      exit(EXIT_FAILURE);
   outfile << N << " " << L << "\n";
}

TxtWriter::~TxtWriter(){
   outfile.close();
}

void TxtWriter::write(Analysis *a){
   outfile << a->getBubbleMeasurement()->getDeltaTime() << " "
         << a->getQuickMeasurement()->getDeltaTime() << "\n";
}

DetailTxtWriter::DetailTxtWriter(unsigned int N, unsigned int L)
: N(N), L(L){
   outfile.open(std::to_string(N) + "_" + std::to_string(L) +
".txt");
   if(!outfile.is_open())
      exit(EXIT_FAILURE);
   outfile << "Размер массива " << N << ".\tКоличество тестов "
<< L << ".\n\n";
}

DetailTxtWriter::~DetailTxtWriter(){
   outfile.close();
}

void DetailTxtWriter::write(Analysis *a){
   outfile << "--------------------["
     << i++
     << '/'
     << L
     << "]--------------------\n"
     << "Сортировка пузырьком.\nВремя(такты): "
     << a->getBubbleMeasurement()->getDeltaTime()
     << "\nВремя(секунды): "
     << a->getBubbleMeasurement()->getDeltaTime()/CLOCKS_PER_SEC
     << "\n----------Массив до сортировки----------\n";
```

```
   printArray(a->getBubbleMeasurement()->getRowArray());
   outfile << "\n----------Отсортированый массив----------\n";
   printArray(a->getBubbleMeasurement()->getArray());
   outfile << "\n-----------------------------------------"
     << "\nБыстрая сортировка Хоара.\nВремя(такты): "
     << a->getQuickMeasurement()->getDeltaTime()
     << "\nВремя(секунды): "
     << a->getQuickMeasurement()->getDeltaTime()/CLOCKS_PER_SEC
     << "\n----------Массив до сортировки----------\n";
   printArray(a->getQuickMeasurement()->getRowArray());
   outfile << "\n----------Отсортированый массив----------\n";
   printArray(a->getQuickMeasurement()->getArray());
   outfile << "\n\n\n\n";
}

void DetailTxtWriter::printArray(int* array){
   int temp = N-1;
   for(int i = 0; i < temp; i++){
      outfile << array[i] << ',';
   }
   outfile << array[temp];
}
```

П1.9 Main.cpp

```
#include <iostream>
#include <conio.h>
#include "Writer.hpp"

//Работа программы с использованием пользовательского интерфейса
в консоли
void uiMod();
void takeAnalysis(unsigned int N,unsigned int L, unsigned int
outformat);

/**
 * Старт программы выполняющей анализ алгоритмов сортировки
 * Если программе передаётся в качестве аргументов
 * размер массива (argv[1]), количество тестов (argv[2]) и фор-
мат вывода (argv[3]),
 * то программа выполняется без вывода/ввода консоли.
 * Во всех остальных случаях программа считывает размер массива
и
 * кол-во тестов с помошью консольного пользовательского интер-
фейса.
 */
int main(int argc, char** argv){
   if(argc != 4)
      uiMod();
   else
      takeAnalysis(atoi(argv[1]), atoi(argv[2]), atoi(argv[3]));
   return 0;
```

```
}
```

```
void uiMod(){
   unsigned int N,L,out_mode;
   std::cout << "Введите размер массива для анализа: ";
   std::cin >> N;
   system("cls");
   std::cout << "Введите количество тестов: ";
   std::cin >> L;
   system("cls");
   std::cout << "Форматы вывода:\n1)JSON\n2)TXT\n3)TXT-
Подробный. Не рекомендуется, для большого кол-ва тестов";
   std::cout << "\nВведите номер предпочтительного варианта: ";
   std::cin >> out_mode;
   system("cls");
   std::cout << "Начат процесс сравнения эффекивности двух алго-
ритмов сортировки.\n"
             << "Выбранные параметры:\nРазмер массива для сорти-
ровки " << N
             << ";\nКоличество тестов " << L << ".\nПодождите
окончания работы программы. "
             << "Если вы ввели большие значения это может занять
более нескольких часов!";
   takeAnalysis(N,L, out_mode);
   system("cls");
   std::cout << "Работа программы успешно завершена!\nНажмите
любую клавишу для выхода";
   getch();
}
#define JSON_WRITER       1
#define TXT_WRITER        2
#define DETAIL_TXT_WRITER 3
void takeAnalysis(unsigned int N,unsigned int L, unsigned int
outformat){
   Writer *w;
   switch(outformat){
      case JSON_WRITER:
         w = new JsonWriter(N,L);
         break;
      case TXT_WRITER:
         w = new TxtWriter(N,L);
         break;
      case DETAIL_TXT_WRITER:
         w = new DetailTxtWriter(N,L);
         break;
      default:
         exit(EXIT_FAILURE);
   }
   for(unsigned int i = 0; i < L; i++){
      Analysis *a = new Analysis(N);
      w->write(a);
      delete a;
   }
   delete w;
```

```
}
```

ПРИЛОЖЕНИЕ 2

Руководство пользователя

Программа работает в двух режимах:

–Получая данные через параметры при вызове в консоли, без UI;

–Получая данные через ввод пользователя в самом приложении, с UI.

Вывод производится в файлы формата “N_L.O”, где N – Размер массива, L – Кол-во тестов, O – Выбранный формат вывода.

Для визуализации результатов работы программы рекомендуется использовать программу GUI.jar

Инструкция по использованию программы с UI.

1)Запустите analysis.exe.

2)Введите размер анализируемого массива.

Рисунок П2.1 Пример ввода размера массива

3)Введите количество тестов.

Рисунок П2.2 Пример ввода кол-ва тестов

4)Выберите формат вывода.

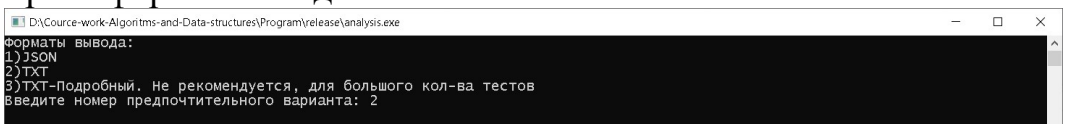

Рисунок П2.3 Пример ввода формата вывода

5)Дождитесь окончания работы программы.

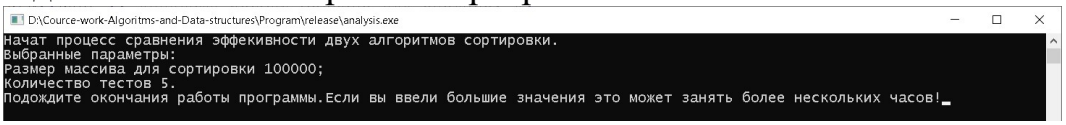

Рисунок П2.5 Скриншот с информацией о работе программы

6)Закройте программу.

Рисунок П2.64 Завершение работы программы

Инструкция по использованию программы без UI.

1)Если используете терминал перейдите в директорию с программой, если .bat файл, создайте его в папке с программой.
2)Используйте данную команду “*analysis N L O*”, где N – размер массива для тестов, L – количество тестов, O – тип вывода (1-.json, 2-.txt, 3-подробный .txt).

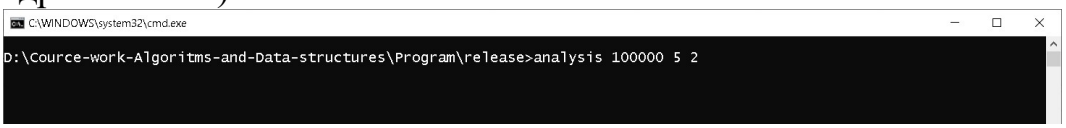

Рисунок П2.75 Пример ввода команды в консоль

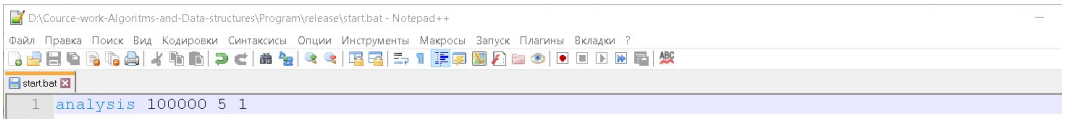

Рисунок П2.8 Пример команды в .bat файле

3)Запустите выполнение в консоли или .bat файле.
4)Дождитесь завершения работы программы.
Инструкция по использованию программы с визуализатором GUI.jar.

1)Запустите программу GUI.jar.
2)В главном меню выберете желаемый режим работы программы.

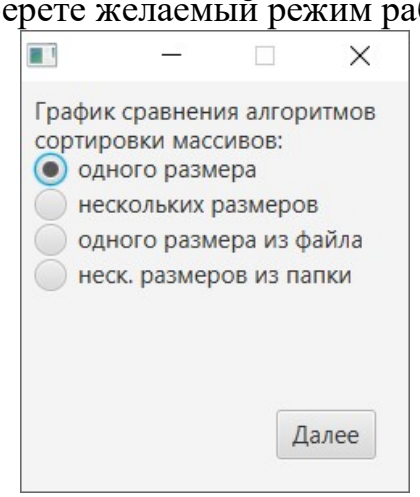

Рисунок П2.96 Главное меню визуализатора

3)Если выбраны первые два варианта, необходимо ввести запрошенные программой данные.

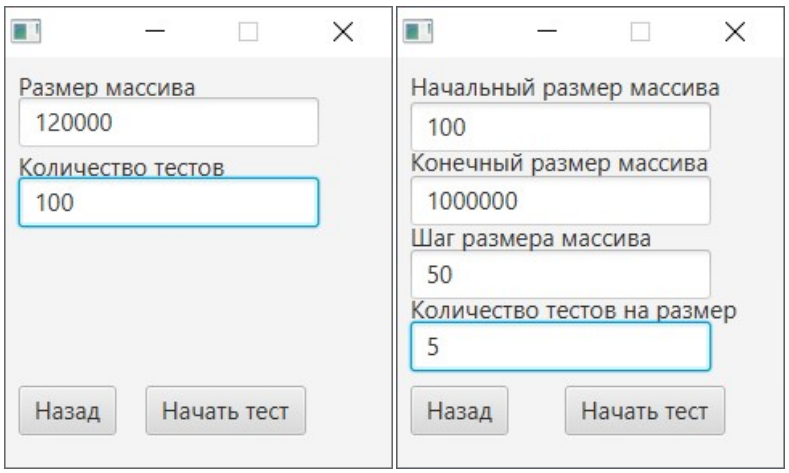

Рисунок П2.10 Меню ввода данных

Если выбраны варианты с выбором файла, откроется файловый менеджер в котором нужно выбрать .json файл или папку с .json файлами с данными об сравнениях.

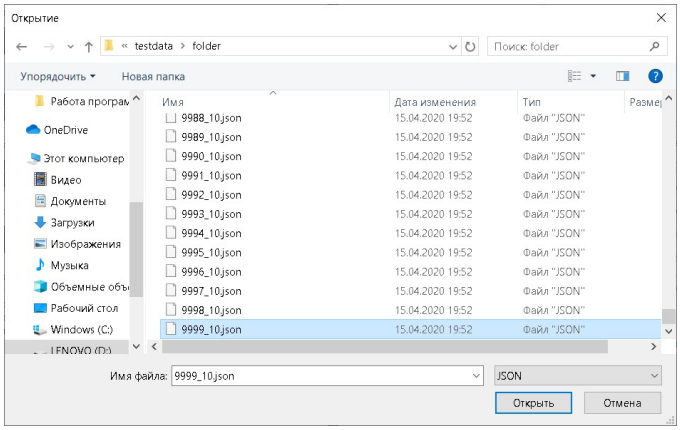

Рисунок П2.11 Выбор файла для построения графика

4)Для первых двух вариантов, необходимо дождаться окончания работы сравнения алгоритмов сортировки. Если был выбран режим сравнения нескольких размеров массивов, будет показан прогресс операции.

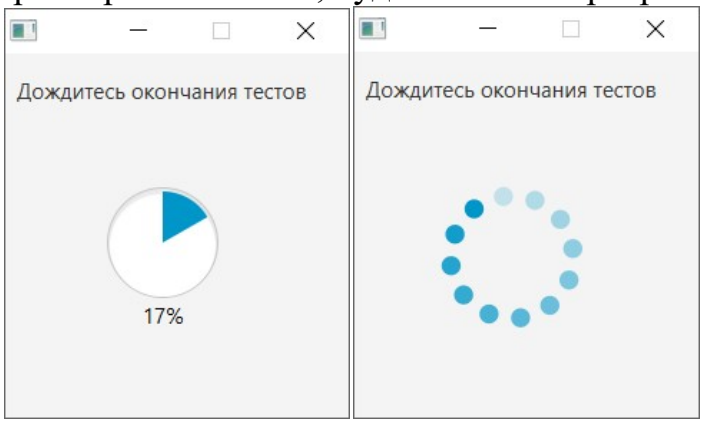

Рисунок П2.12 Ожидание окончания работы программы

5)График построен, есть возможность распечатать его или сохранить в качестве изображения.

Рисунок П2.13 График визуализирующий данные сравнения алгоритмов

ПРИЛОЖЕНИЕ 3

Результат выполнения программы

Вид записи результатов выполнения программы зависят от выбранного пользователем режима вывода. Время выполнения во всех режимах вывода представлено в тиках (процессорное время). Название файла, исключая расширение, содержит размер массива и количество тестов.

Вывод в .txt

Первая строка содержит информацию о размере массива и количестве тестов. Последующие строки содержат время выполнения алгоритмов. Первым идёт время выполнения метода пузырька, следующим время выполнения быстрой сортировки Хоара.

```
100000 5
19688 11
24658 10
22593 10
20224 11
19367 14
```

Вывод в .json

N содержит информацию о размере массива, L содержит информацию о количестве тестов, массив Measurements содержит объекты с информацией о тестах, где bubble время выполнения метода пузырька, quick а время выполнения быстрой сортировки Хоара

```
{
   "N" : 100000,
   "L" : 5,
   "Measurements": [
      { "bubble" : 17470, "quick" : 9},
      { "bubble" : 18174, "quick" : 9},
      { "bubble" : 31419, "quick" : 14},
      { "bubble" : 21473, "quick" : 10},
      { "bubble" : 17495, "quick" : 10}
   ]
}
```